

# Challenges in Congestion Control in the Internet

Jorge Gonçalves  
MIT, LIDS

May 30th, 2000



MIT

EECS, LIDS

## Internet

- The rather decentralized and fast-changing evolution of the Internet architecture has worked reasonably well to date.

**There is no guarantee it will continue to do so**

Area Exam, MIT, LIDS

3



MIT

EECS, LIDS

## Internet

- Enormous Network with many different service providers, users, protocols, buffer memory at gateways, link's bandwidth, processor speed...
- No central control or authority with many players independently making changes
- Doubling its traffic every few months
- Increasing number of non-adaptive flows

Area Exam, MIT, LIDS

1



MIT

EECS, LIDS

## Internet

- The rather decentralized and fast-changing evolution of the Internet architecture has worked reasonably well to date.

**There is no guarantee it will continue to do so**



Internet is vulnerable and at high risk of frequent **congestion** and **collapses**

Area Exam, MIT, LIDS

4



MIT

EECS, LIDS

## Internet

- The rather decentralized and fast-changing evolution of the Internet architecture has worked reasonably well to date.

Area Exam, MIT, LIDS

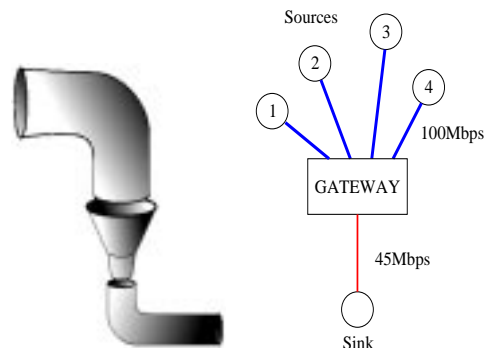
2



MIT

EECS, LIDS

## Congestion Control



**Congestion:**

$$\sum \text{Demand} > \text{Available Resources} \Rightarrow \text{Packet Loss}$$

Area Exam, MIT, LIDS

5

**Congestion Control**

Goals of Congestion Control:

- Maximize throughput
- Minimize the average delay of packets in the network
- Fair allocation of resources among all users

**TCP—Transmission Control Protocol**

Introduced in the mid' 80s to provide end-to-end congestion control.

Goals:

- Decrease delays
- Increase throughput
- Introduce fairness
- Avoid congestion collapses
- Make the Internet **Stable** and **Robust**

≈ 95% of today's flows are TCP.

**Congestion Control**

Goals of Congestion Control:

- Maximize throughput
- Minimize the average delay of packets in the network
- Fair allocation of resources among all users

**Congestion** cannot be solved by just increasing memory, bandwidth, processor speed. In fact, this may lead to even more congestion.

**TCP**

How it works:

- A TCP flow increases its sending rate of packets as long as no packets are lost
- Detection of packet loss—Congestion
- TCP decreases the sending rate

The sending rate of adaptive applications is changed according to the level of congestion perceived in the network.

TCP adjusts the long term transmission rate without any explicit feedback from the network.

**Congestion Control**

More requirements a congestion control scheme must satisfy:

- **Low overhead.** In particular, it should not increase traffic during congestion. This is one of the reasons why explicit feedback messages are considered undesirable.
- **Responsive.** The congestion control scheme is required to match the demand dynamically to the available capacity.
- **Must** continue to **work** even when, the rate of transmission errors, out-of-sequence packets, deadlocks, and lost packets increases considerable under congestion.

**TCP**

TCP by itself is not enough!

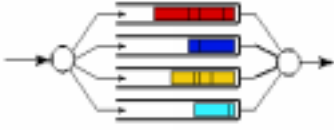
- Queues build up and packet drop occurs only when the queues are full ⇒ **increase delays**
- Global synchronization— all flows reduce their sending rate simultaneously ⇒ **decrease in throughput**
- Some flows are not TCP compliant—may not respond to congestion ⇒ **will take over links' bandwidth**

Need to control congestion at gateways as well.

**Congestion Control at Gateways**

Two types of algorithms:

- Keep separate queue for each flow

**Scheduling Algorithms****Advantages**

- A Flow cannot degrade the quality of other flows
- Gives a fair share of bandwidth to all users

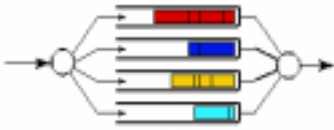
**Problems**

- Does not scale well to a large number of users
- Requires heavy and expensive computations
- Unrealistic in today's high-speed links—rarely used in networks

**Congestion Control at Gateways**

Two types of algorithms:

- Keep separate queue for each flow

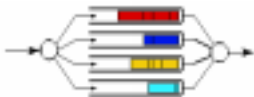


- Use a single FIFO queue

**Queue Management Algorithms**

FIFO—First In First Out

- Single FIFO packet queue shared by all flows
- An algorithm manages the length of the packet queue by dropping packets when necessary or appropriate.

**Scheduling Algorithms****Advantages**

- A Flow cannot degrade the quality of other flows
- Gives a fair share of bandwidth to all users

**Queue Management Algorithms**

FIFO—First In First Out

- Single FIFO packet queue shared by all flows
- An algorithm manages the length of the packet queue by dropping packets when necessary or appropriate.

**Advantages**

- Require none (or at least much less) state information
- Scales well

**Drop Tail**

The most simple gateway algorithm. Drop the arriving packet if the buffer is full.

**Advantages**

- Simple

**Early Random Drop**

**Idea:** prevent congestion, rather than just react to it.

**How:** drop packets before the gateway's buffers are completely exhausted.

**Algorithm:** If the queue length exceeds a certain drop level, drop each packet arriving at the gateway with a fixed drop probability.

- Improvement over Random Drop and Drop Tail but still with many problems.
- Need better congestion algorithms.
- Tradeoff: Computation vs Efficiency

**Drop Tail**

The most simple gateway algorithm. Drop the arriving packet if the buffer is full.

**Advantages**

- Simple

**Disadvantages**

- If buffer is long, a packet may experience long delays
- If buffer is short, buffer does not accommodate bursty traffic
- Global synchronization can happen  $\Rightarrow$  loss of throughput
- May lead to network collapses

**RED—Random Early Drop**

**Fact:** real traffic is very bursty

**Idea of RED:** keep the average queue size low.

**How:** drop packets when the average queue length is high.

**Algorithm:**

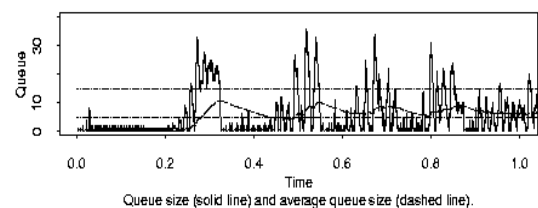
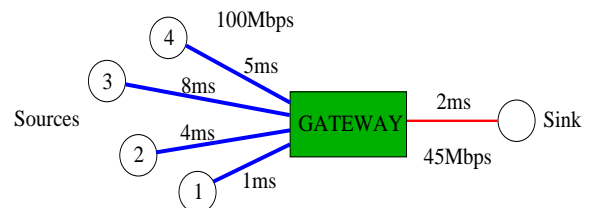
```

for each packet arrival
  calculate the new average queue size avg
  if  $min_{th} \leq avg < max_{th}$ 
    calculate probability  $p_a$ 
    with probability  $p_a$ :
      drop the arriving packet
  else if  $avg \geq max_{th}$ 
    drop the arriving packet
  
```

**Random Drop**

For each arriving packet, if the buffer is full, randomly choose a packet from the gateway queue to drop.

- Still simple but with the same problems as Drop Tail

**RED—Simulations**

**RED—Advantages**

- Better performance than Drop Tail: higher throughput, lower delays
- Avoids global synchronization
- Accommodation of short bursts
- The probability that a packet is dropped is proportional to that connection's share of the throughput through the gateway.
- Control of the average queue size, even in the absence of non-adaptive sources
- Easy to implement
- In 1998, RED has been recommended by a handful of researchers as the standard of congestion avoidance mechanism in gateways

**RED—Problems**

- Dropping packets from flows in proportion to their bandwidths does not always lead to fair bandwidth sharing
- RED is designed to work with adaptive flows. Non-adaptive flows can take over the link's bandwidth
- RED heavily penalizes TCP flows and "rewards" non-TCP flows
- Difficulty in setting parameters

**RED—Problems**

Simulation of 32 TCP sources and 1 UDP source. The link's bandwidth is 1Mbps. The UDP sends packets at a rate of 2Mbps.

**FRED—Flow Random Early Drop**

- Modified version of RED

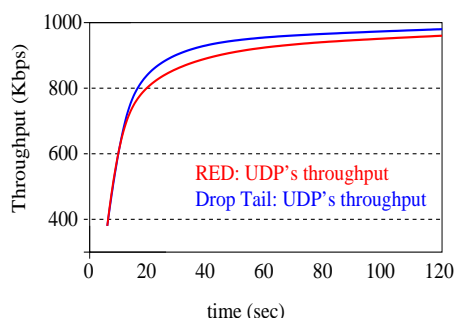
**Goal:** provide better protection for adaptive flows and isolate non-adaptive greedy traffic.

**Idea:** uses per-active-flow accounting to impose on each flow a loss rate that depends on the flow's buffer use.

**How:** maintain state information of all flows currently present in the gateway.

**RED—Problems**

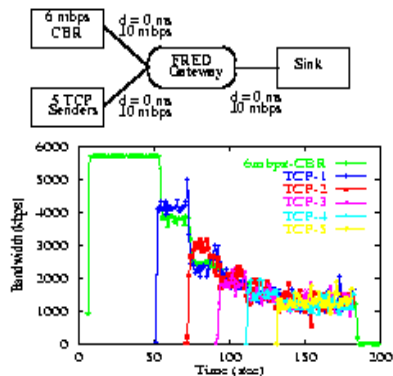
Simulation of 32 TCP sources and 1 UDP source. The link's bandwidth is 1Mbps. The UDP sends packets at a rate of 2Mbps.

**FRED—Algorithm**

FRED acts just like RED, with the following differences

	RED	FRED
$Q_{ave}$		
$max_{th}$	Drop	Drop
	Random Drop	R2: Drop if $Q_i > Q_{ave}/N$ R1: Accept if $Q_i < Q_{ave}/N$ else random drop
$min_{th}$		
0	Accept	R2: Drop if $Q_i > Q_{ave}/N$ else accept

- For each flow, FRED counts the number of times the flow has failed to respond to congestion notification (*strike*)

**FRED—Simulations**

UDP flow does gets approximately its fair share of bandwidth.

**SRED—Stabilized Random Early Drop**

**Goal:** identify flows that are taking more than their fair share of bandwidth, and to allocate a fair share of bandwidth to all flows without incurring in too many computations.

**Idea:** estimate the number of active flows in the buffer.

**How:** keep a **small list** where flows with high bandwidth are likely to be in the list.

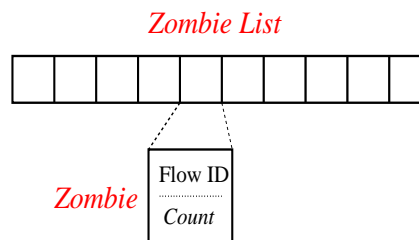
The packet drop probability depends only on the instantaneous buffer occupation and on the estimated number of active flows.

**FRED—Advantages**

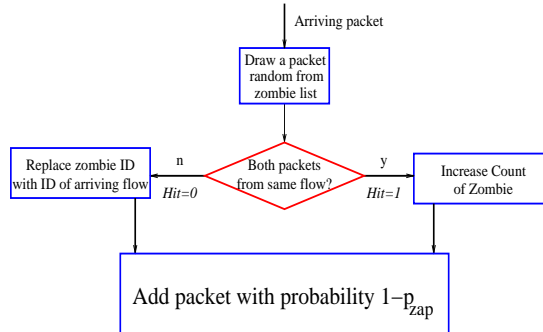
- Brings **fairness** to all users. Big improvement over RED
- Flows with fewer packets queued in the buffer than this average are favored over flows with more

**SRED—Algorithm**

- The **Zombie List** keeps a small list of recently seen flows together with a variable **Count** for each flow in the list

**FRED—Problems**

- **Computationally TOO expensive!** Needs state information about all active connections
- **Does not scale**—uses memory in proportion to the total number of buffers used.
- Setting parameters

**SRED—Algorithm**

**SRED—Algorithm**Drop probability  $p_{zap}$ 

$$P[n] = (1 - \alpha)P[n - 1] + \alpha Hit[n]$$

**SRED—Problems**

- Computationally is still too expensive for some high-speed links
- Needs further research to control non-adaptive flows.
- Estimation of the number of active flows is not accurate when files have random sizes, rather than infinite size.
- Setting parameters

**SRED—Algorithm**Drop probability  $p_{zap}$ 

$$P[n] = (1 - \alpha)P[n - 1] + \alpha Hit[n]$$

$$p_{sred}(q) = \begin{cases} p_{max} & \text{if } B/3 \leq q < B \\ p_{max}/4 & \text{if } B/6 \leq q < B/3 \\ 0 & \text{if } 0 \leq q < B/6 \end{cases}$$

$$p_{zap} = p_{sred}(q) \times \min \left\{ 1, \frac{1}{256 P^2[n]} \right\} \times \left( 1 + \frac{Hit[n]}{P[n]} \right)$$

**CHOKe**

- Modified version of RED

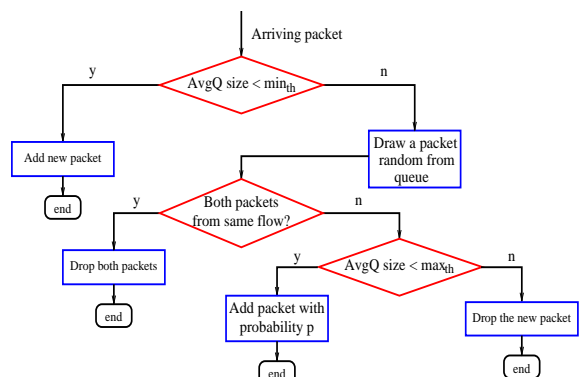
**Goal:** **simultaneously identify and penalize** misbehaving flows with simple implementation while preserving the positive key features of RED.

**Idea:** identify and reduce the allocation of the flows which consume the most resources, i.e., attempt to minimize the resource consumption of the maximum flow.

**How:** compare the arriving packet with a randomly chosen packet from the buffer queue, and possible drop one or both packets.

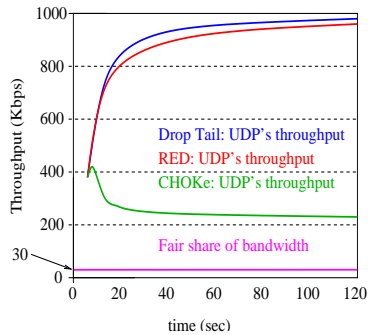
**SRED—Advantages**

- Stabilizes the buffer at a level independent of the number of active connections
- The estimate of the number of active flows in the buffer is obtained without collecting or analyzing state information on individual flows
- SRED does not compute the average queue length

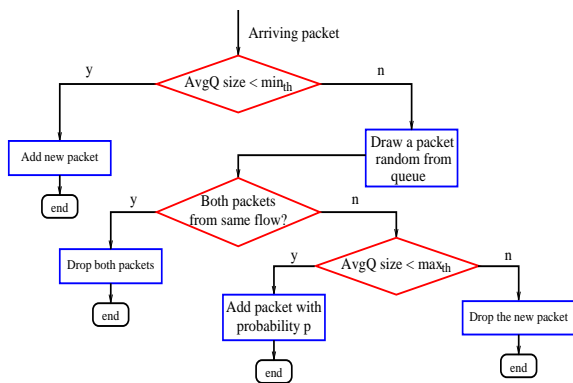
**CHOKe—Algorithm**

**CHOKE—Simulations**

Back to simulation with 32 TCP sources and 1 UDP source. Link's bandwidth: 1Mbps. The UDP sends packets at a rate of 2Mbps.

**CHOKE—Advantages**

- Stateless, simple, and easy to implement
- Preserves key features that RED possesses
  - ability to avoid global synchronization
  - keep buffer occupancies small and ensure low delays
  - lack of bias against bursty traffic
- Unlike RED penalizes unresponsive flows

**CHOKE—Algorithm**

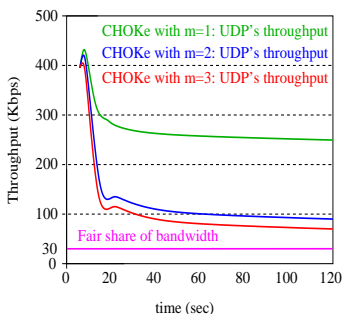
- Can compare with  $m$  packets instead of just 1

**CHOKE—Problems**

- With  $m = 1$ , UDP flow still has a throughput 10 times higher than its fair share  $\Rightarrow$  CHOKE “rewards” UDP flows
- Even with higher  $m$  (heavier computations) CHOKE still “rewards” UDP flows
- Multiple UDP flows take all the bandwidth when  $m = 1$
- Large values of  $m$  make CHOKE an expensive algorithm
- No real incentives are given by the gateway to use TCP. In fact, there are even “rewards” for not being TCP compliant

**CHOKE—Simulations**

Same simulation for  $m = 1, 2, 3$

**Discussion**

Efficient:

- Scheduling algorithms
- FRED
- SRED?
- CHOKE with large  $m$ ?

**Discussion****Efficient:**

- Scheduling algorithms
- FRED
- SRED?
- CHOKe with large  $m$ ?

**Simple:**

- Drop Tail, Random Drop, Early Random Drop
- RED
- CHOKe with small  $m$

**Discussion****Danger in today's Internet:**

- Software vendors are taking advantage of TCP
- Their products are not TCP compliant and are aggressive
- They take over link's bandwidth

**Discussion—Why incentive TCP?**

Internet is an enormous network with

- millions of different users
- different service providers
- different link speeds, buffer space, processor time, etc.
- different protocols

**Discussion****Danger in today's Internet:**

- Software vendors are taking advantage of TCP
- Their products are not TCP compliant and are aggressive
- They take over link's bandwidth

**Solution:**

- Give concrete incentives at the gateway to connections to use TCP
- How: identify and heavily drop packets from non-adaptive flows

Right now there are no real incentives for connections to be TCP!

**Discussion—Why incentive TCP?**

Internet is an enormous network with

- millions of different users
- different service providers
- different link speeds, buffer space, processor time, etc.
- different protocols

TCP has made the Internet

- Stable, robust
- Scales well
- Conservative
- **IT WORKS!**

**Proposed Improvements**

Divided in two parts

- Identify non-adaptive flows
- Give only small fractions of the bandwidth to such flows by heavily dropping their packets

**Identify non-adaptive flows**

**Idea:** estimate the arrival rate of high-bandwidth flows from recent packet drop history at the gateway.

**How:** keep a short list of packets that were recently dropped.

- Every time a packet is dropped, compare it with a random packet from the list.
- If they match, increase its *Count*
- Otherwise, the dropped packet replaces the randomly chosen packet, and *Count* is set to one
- Let  $C_i$  be the sum of *Count* over all elements in the list from flow  $i$

**Penalize non-adaptive flows**

Two things can happen

- The connection is TCP with a large round-trip time

**Identify non-adaptive flows**

- Compare  $C_i$  with  $C_j$ , all  $j \neq i$
- Or, compare  $C_i$  with the average over all  $C_j$

Non-adaptive flows do not reduce their sending rate.

**Penalize non-adaptive flows**

Two things can happen

- The connection is TCP with a large round-trip time
- The connection is non-adaptive

**Penalize non-adaptive flows**

- When a packet arrives at the gateway, proceed as in RED
- If in random drop, the drop probability is as in RED except when
  - $C_i$  is higher than some threshold
  - or  $C_i$  is much higher than all other  $C_j$
  - or  $C_i$  is higher than the average over all  $C_j$
- In this case, the drop probability should be very high

**Conclusions**

- This algorithm incentives end-to-end congestion control
- It can certainly be further improved
- A number of algorithms to incentive the use of TCP will emerge in a near future
- Such algorithms should then be implemented as soon as possible